N J I T
New Jersey's Science & Technology University
THE EDGE IN KNOWLEDGE

CS 280 Programming Language Concepts

classes

NJIT New Jersey's Science & Technology University
COLLEGE OF COMPUTING SCIENCES

Classes: Java vs. C++

- Many similarities, some differences
- Java and C++ indicate visibility of class members slightly differently
- C++ methods do not have to be defined in the body of the class
- C++ uses “this” as a pointer to the current object
- Java uses references exclusively, so “this” is a reference to the current object

NJIT New Jersey's Science & Technology University — COLLEGE OF COMPUTING SCIENCES

Class Definitions

```
//JAVA
class MyClass {
   private int a,b;

   public MyClass() {
       a = b = 0;
   }

   public int getA() {
       return a;
   }
}
```

```
//C++
class MyClass {
private:
   int a,b;

public:
   MyClass() {
       a = b = 0;
   }

   int getA() {
       return a;
   }
}; // notice the ;
```

NJIT New Jersey's Science & Technology University — COLLEGE OF COMPUTING SCIENCES

Terminology

- The class definition lays out the “members” and the “member functions” or “methods” for the class
- Think of the class as a type with behaviors (methods) that are defined by the programmer
- An object is an instance of the class that is created by the programmer
- There can be many instances of the class

Constructors

- A member function whose name is the class name is a “constructor”. It’s called when storage is bound to an object
- There can be many constructors for a class, each with a different calling sequence for the function

Public and Private

- A public member can be seen by anyone
- A public method can be called by anyone
- A private member can only be seen inside the class (by other class members)
- A private method can only be called from inside the class (by other methods)

Static

- Static works the same in C++ and Java
  - Static members have a single memory binding set at load time
  - Static methods can only access static members
  - Static members and methods are accessed by way of the class name
- In C++, static members must be declared and initialized somewhere outside the class definition

Defining functions inside vs. outside

```
//C++
class MyClass {
private:
    int a,b;

public:
    MyClass() {
        a = b = 0;
    }

    int getA() {
        return a;
    }
};

// methods defined inside
// a class definition might
// be "inlined"
```

```
//C++
class MyClass {
private:
    static int x;
    int a,b;
public:
    MyClass();
    int getA();
};


// below can be in a separate file
int MyClass::x = 0;

MyClass::MyClass() {
        a = b = 0;
}

int MyClass::getA() {
        return a;
}
```

NJIT New Jersey's Science & Technology University   COLLEGE OF COMPUTING SCIENCES

Initializers

- C++ lets you initialize class members with initializer lists outside the constructor
- These three do the same thing:

```
MyClass() { a = b = 0; }
MyClass() : a(0) { b = 0; }
MyClass() : a(0), b(0) {}
```

- This syntax can be used to construct any parent classes as well

NJIT New Jersey's Science & Technology University   COLLEGE OF COMPUTING SCIENCES

Default Parameters

- C++ lets you provide default values for function parameters

```
MyClass(int a, int b=0) : a(a), b(b) {}
```

  - The constructor requires 1 or 2 parameters
  - If a second parameter is not provided, it takes the default value (0 in this case)
  - You can have many default parameter values but they must be the rightmost parameters
  - Note the {} – the body of this constructor's empty because all the initialization has been done
  - Note the syntax a(a): those a's have different scopes!
  - Note if this is the only constructor defined then you MUST provide at least one parameter. In this example, MyClass() is invalid

Destructors

- C++ allows the programmer to define a destructor
- A destructor is a function whose name is a tilde (~) followed by the class name
- There can be at most one destructor; if it exists it is called when memory is unbound from the variable
- Destructors are called automatically and cannot be called directly by the programmer

Object Creation

- Declaring a variable to have the type of a given class is the same as declaring a variable of any other type:
  - When the variable has memory assigned to it, any constructors are called
  - When the variable's address is unbound, the destructor, if any, is called
- Note this is DIFFERENT from Java, where the variable is a reference to a class. This reference must be initialized with new.

Copying Objects

- C++ will copy objects if you assign one to the other.
- This is a use of assignment: operator=
- By default, operator= simply copies each member, one member at a time
- You can create your own:

```
MyClass& operator=(const MyClass& rhs);
```

- Last line of this function is `return *this;`
- If you don't want your object copied… make your operator= private

Constructing by Copying Objects

```
MyClass c1;
MyClass c2(c1);
```

- This calls a “copy constructor” that the compiler generates for you by default
- A copy constructor just copies the members from the source to the destination

- C++ allows you to control this by defining your own copy constructor
- The following defines a copy constructor to replace the default compiler-generated one:

```
MyClass(const MyClass& copyfrom){ … }
```

Accessing Members

- Accessing members looks the same in Java and C++

```
MyClass obj;
// in Java I’d need code here to set obj
int x = obj.getA()
```

- The . (dot) operator accesses a member
- Note in Java everything is a reference so obj is a reference and obj. follows the reference to access the member that the reference refers to

Heap Allocated Objects in C++

```
MyClass *cp; // pointer to a MyClass

cp = new MyClass(); // allocates

int x = cp->getA(); // note ->

delete cp; // de-allocates
```

NJIT New Jersey's Science & Technology University COLLEGE OF COMPUTING SCIENCES

Printing Your Object

- Java allows you to specify how an object is printed out by letting you define a toString() method
- In C++, we want to be able to print an object to an iostream, the same way that we can print numbers and strings
- Since iostream works by overloading the << operator, we need to define how << works for our object

NJIT New Jersey's Science & Technology University COLLEGE OF COMPUTING SCIENCES

Iostream details

- iostream works by overloading the << operator
- When you write code that says, for example, `cout << 10;` the compiler generates a call to the function `operator<<(cout, 10);`
- To make your class printable on an iostream you must define your own overloaded operator<< function

```
//C++
class MyClass {
private:
    int a,b;

public:
    MyClass() {
        a = b = 0;
    }

    int getA() {
        return a;
    }

    friend ostream& operator<<(ostream& out, const MyClass& o);
};
```

A “friend” function is a function that is not actually a member of your class, but that is allowed to have access to private members of your class

```
// function must return the first argument
// so that a sequence of << operators will work

ostream& operator<<(ostream& out, const MyClass& o)
{
     // format output any way you like, and
     // send it to "out"
     out << "(a,b=" << o.a << "," << o.b << ")";
     return out;
}
```

Inheritance

- Java
  - class B extends A {}
  - java supports implementing interfaces:
    - class B extends A implements I1, I2, I3 {}
- C++
  - class B : public A {}
  - Multiple inheritance is allowed:
    - class C : public B, public D {}

Inheritance

- Classes inherit things from their parents: members, methods, etc.
- In concept, a child "is a" instance of the parent
- If a parent has a method x(), then the child has that method, too
- In object oriented programming, we control inheritance, and we can change how methods work inside of the children

NJIT New Jersey's Science & Technology University COLLEGE OF COMPUTING SCIENCES

Virtual Functions

- A function defined in a base class can be overridden in a derived class by simply providing a new implementation with the same argument signature
- You may decide to declare that the function in the base class is "virtual"
- When a function is virtual, the proper function to be used is determined at run time
- THEREFORE in C++ some functions are bound at compile time and some are bound at run time
- Java does not have virtual functions, and ALWAYS determines the proper function to call at run time
- In Java you can say that a function cannot be redefined by declaring the function final

NJIT New Jersey's Science & Technology University COLLEGE OF COMPUTING SCIENCES

NJIT

THE EDGE IN KNOWLEDGE